

# Simple Job Cycle

Last review date	Reviewer
2009-09-15	Marco Bencivenni Enrico Fattibene

## Table of Contents

### [Simple Job Cycle](#)

[Job commands](#)

[Proxy Delegation](#)

[Preparing a Simple Job](#)

[Other important attributes](#)

[The "Hello World" job](#)

[Submit a job](#)

[The status of a job](#)

[Recovering Results](#)

[Retrieving job history](#)

[Job- compatible Resources](#)

## Simple Job Cycle

An explanation of how to use the job management commands to prepare and submit a simple job, monitor its status and retrieve the output.

### Job commands

Most jobs are submitted to the EGEE Grid infrastructure through the gLite WMS (Workload Management System). This is a Grid meta-scheduling service that matches job requirements to the capabilities of resources and chooses the most appropriate resource for a particular job. An older meta- scheduling service called the LCG RB (Resource Broker) is deprecated; the commands for that service are not covered here.

The commands used for handling jobs on the Grid are:

Submitting a job	<code>glite-wms-job-submit</code>
Checking job status	<code>glite-wms-job-status</code>
Retrieving job output	<code>glite-wms-job-output</code>
Checking job history	<code>glite-wms-job-logging-info</code>
Listing compatible resources	<code>glite-wms-job-list-match</code>
Delegate a proxy	<code>glite-wms-job-delegate-proxy</code>

All job management commands require the user to have a valid proxy to work, as all interactions with the WMS are secure and authenticated.

### Proxy Delegation

The gLite WMS allows more flexible proxy delegation than previous meta- scheduling services, requires explicit delegation of a proxy. Consequently, the gLite WMS has an additional command to do this delegation (`glite-wms-job-delegate-proxy`) and required delegation parameters for the `glite-wms-job-submit` and `glite-wms-job-list-match` commands.

### Automatic Delegation

To use automatic delegation, one simply needs to add the `-a` option when using the `glite-wms-job-submit` and `glite-wms-job-list-match` commands. This will transmit your active proxy to the WMS system and use it for the job.

## Manual Delegation

You may want to use different proxies for different classes of jobs to allow those jobs to have different authorizations (e.g. via the use of VOMS groups or roles). To manually delegate a proxy, create a local proxy with the groups and roles you want with the `voms-proxy-init` command. To delegate this proxy, do the following:

```
glite-wms-job-delegate-proxy -d myproxy
```

Replace "myproxy" with a descriptive name of your choice. You can have many different, named proxies delegated to the WMS system. This proxy can then be used for a particular job submission via the `-d` option. For example,

```
glite-wms-job-submit -d myproxy MyJob.jdl
```

This will use the proxy identified by the name "myproxy" for the submitted job.

## Preparing a Simple Job

The main purpose of the Grid is to allow people to run their programs in the most efficient way. In Grid terminology, a job is a unit of work intended as a program which starts, reads some data, does some calculation, produces an output and finishes. A job is described by a file, expressed in a language called Job Description Language, often referred to as a JDL file. A JDL file contains information such as

- the name of the program to run;
- the input files read by the program;
- the output files produced by the program;
- the requirements to be satisfied by the host which is going to execute the program.

The Grid is made up, among other things, of *Computing Elements (CE)*, which correspond physically to clusters of computers located in a computer centre. Users submit their jobs to the *WMS*, which looks at the corresponding JDL files and dispatches the jobs to the best available resources satisfying the job requirements, where the adequacy of a resource is measured by an estimate of the time to wait from job submission to job execution (which can be large, in case the resource has already a long queue of jobs to process).

## The Job Description Language

The Job Description Language is a high-level language used to describe jobs. A JDL file is a text file containing a series of key-value pairs with the format:

```
attribute = expression;
```

where every pair is terminated by a semi-colon and can span several lines. Comments in a JDL file must be preceded by a pound (#) or enclosed between `/*` and `*/`.

In the following examples, the most important JDL attributes will be explained.

### The simplest possible job

The most basic example of a Grid job consists in executing a simple command and retrieving its output. This JDL file shows code to do this:

```
# example.jdl
Executable = "/bin/hostname";
StdOutput  = "std.out";
StdError   = "std.err";
```

The `Executable` attribute specifies the command to run in the job. The `StdOutput` attribute specifies the file where the standard output of the job will be written, and similarly the `StdError` attribute specifies the file where the standard error of the job will be written.

The above JDL file does not allow to get back the output of the job. This is done via the concept of *sandbox*.

## The sandbox

The sandbox mechanism allows users to specify what files should be sent together with the job from the UI to the execution host, and sent back from to execution host to UI upon successful completion of the job. This is achieved by using the `InputSandbox` and the `OutputSandbox` attributes, respectively.

The `InputSandbox` attribute contains typically the executable to be run, when it is not already present on the execution host, and any other file possibly needed for the executable to run and which is located on the User Interface. For example:

```
InputSandbox = {"/home/does/test.sh", "fileA", "fileB"};
```

Relative paths are relative to the current directory when the user submits the job to the Grid. Wildcards (the `*` character) can be used, but it is forbidden to specify two or more files with the same file name, even if their paths are different.

The `OutputSandbox` attribute contains the files for the standard output and the standard error, plus any other file created by the job executable that the user wants to keep. For example:

```
OutputSandbox = {"std.out", "std.err", "output.data"};
```

All the files in the output sandbox must be expressed as relative paths. Their absolute paths cannot be known in advance, as the jobs are executed in temporary directories on the execution host.

It is very important that the files in the sandboxes are limited both in size and in number. This is because every sandbox file has to be separately transferred from the UI to the WMS and then to the execution host, or vice versa; jobs having a lot of files or very big files in the sandboxes have an impact on the performance of the WMS.

## Other important attributes

The `Arguments` attribute can be used to specify command- line arguments for the command specified in the `Executable` attribute. For example:

```
Executable = "/bin/echo";  
Arguments = "Hello world!";
```

Special characters such `"`, `&`, `|`, `\`, `<`, `>` must be escaped by preceding them with a `\`. This means that the parser which processes the JDL commands knows when to regard a special character as part of a string, as in other parsers.

## The "Hello World" job

The following job executes a command to print the string "Hello world!" to the standard output. The executable is called `test.sh` and is a shell script with execution flag enabled:

```
#!/bin/sh  
# test.sh  
echo $*
```

while the JDL file is:

```
# test.jdl  
  
Executable      = "test.sh";  
Arguments       = "Hello world!";  
StdOutput       = "std.out";  
StdError        = "std.err";  
InputSandbox    = {"test.sh"};  
OutputSandbox   = {"std.out", "std.err"};
```

## Submit a job

The submission of a job is accomplished by executing the command:

```
$ glite-wms-job-submit [-o out_file] <jdl>
```

If successful, the command returns a string which identifies the job from that moment on (referred to as the *jobId*). The *jobId* has the format:

```
https://<wmpoxy>[:port]/unique_string
```

where *<wmpoxy>* is the host name of the WMPProxy server, usually the machine to which the job was submitted.

The optional `-o <out_file>` appends the *jobId* to a file, which is rather convenient way to keep track of the submitted jobs.

It is worth to noting that the WMS server chosen to submit the job depends on the configuration of the User Interface.

### Example: simple job submission

You can try now to submit the job described by the `test.sh` and `test.jdl`, with the following command:

```
$ glite-wms-job-submit -a test.jdl
```

You will get an output similar to:

```
Connecting to the service https://prod-wms-01.pd.infn.it:7443/glite_wms_wmproxy_server
===== glite-wms-job-submit Success =====
The job has been successfully submitted to the WMPProxy
Your job identifier is:
https://prod-wms-01.pd.infn.it:9000/0rbdA50itN26nZJJEggXnw
=====
```

if the job submission was successful.

## The status of a job

The status of a job can be obtained using the command:

```
$ glite-wms-job-status <jobId>
```

or:

```
$ glite-wms-job-status -i <file_path>
```

where *<jobId>* and *<file\_path>* are, respectively, a *jobId* and a file containing a list of *jobIds* (like the file produced by the `-o` option of `glite-wms-job-submit`).

There are many possible states a job may find itself in. The following table has a brief description of their meaning.

Status	Description
Submitted	The job has been accepted by the grid
Waiting	An appropriate resource has yet to be selected
Ready	An appropriate resource has been selected
Scheduled	The job has been correctly submitted to the remote resource
Running	The job is currently running
Done	The job has finished (either correctly or not)
Cleared	The job's output has been retrieved
Aborted	The job failed and has been terminated
Cancelled	The job has been cancelled

The correct sequence of states for a good job is 'Submitted'-'Waiting'-'Ready'-'Scheduled'-'Running'-'Done'. Due to the internal latencies of the system, even for the shortest job, a few minutes will be needed to see 'Done' status and be able to recover the output.

It is impossible to see the status of another users' jobs.

### Example: Checking the status of a job

This command retrieves the job status of the previously submitted job, in section Submit a job:

```
$ glite-wms-job-status https://prod-wms-01.pd.infn.it:9000/KJ3A7ooH3nNUK5M1jzBLuA
```

If the job ended correctly, the result will be similar to:

```
*****
BOOKKEEPING INFORMATION:

Status info for the Job : https://prod-wms-01.pd.infn.it:9000/KJ3A7ooH3nNUK5M1jzBLuA
Current Status:      Done (Success)
*****
```

## Recovering Results

The 'standard' job result retrieval command is:

```
$ glite-wms-job-output < job identifier >
```

where the `job identifier` is the job identifier string returned from the `glite-wms-job-submit` command above. The result of the `glite-wms-job-status` command for this job must indicate that the status of the job is "Done" before results can be retrieved.

For this example the job result recovery command is:

```
glite-wms-job-output https://prod-wms-01.pd.infn.it:9000/fHJ43bqB-q_9gIGILxDtBg
```

After issuing this command, you should see:

```
Connecting to the service https://193.206.210.111:7443/glite_wms_wmproxy_server
=====
                        JOB GET OUTPUT OUTCOME
=====

Output sandbox files for the job:
https://prod-wms-01.pd.infn.it:9000/fHJ43bqB-q_9gIGILxDtBg
have been successfully retrieved and stored in the directory:
/grid/users/loomis/JobOutput/loomis_fHJ43bqB-q_9gIGILxDtBg
=====
```

As a default, the output of the job is recovered to your home machine under the `/tmp` directory in a directory named by the scheme `< user id >_< job id >`. In this example, the output directory was overridden by a local configuration file that specified the "OutputStorage" parameter to be `"/grid/users/loomis/JobOutput"`. It could equally as well been specified using the `--dir` option.

## Retrieving job history

The commands used to retrieve the history of a job are:

```
See job history | glite-wms-job-logging-info
```

All job management commands require the user to have a valid proxy to work, as all interactions with the WMS are secure and authenticated.

### Retrieve the history of a job

Sometimes, it is useful to know the history of a job through the WMS from submission to end, for example to better understand the reason of a failure, when `glite-wms-job-status` is not enough. The history, also called *logging information* of a job can be retrieved by doing:

```
$ glite-wms-job-logging-info [-v <verbosity>] <jobId>
```

or

```
$ glite-wms-job-logging-info [-v <verbosity>] -i <file_path>
```

where `<jobId>` and `<file_path>` are respectively a jobId and a file containing a list of jobIds (like the file produced by the `-o` option of `glite-wms-job-submit`). The optional parameter `-v <verbosity>` is used to set the amount of detail in the output, and it is advisable to use `-v 3` to get the most complete information.

## Job-compatible Resources

It is possible to see which Computing Element (CE) satisfies the requirements for a job by using the command

```
$ glite-wms-job-list-match -a [--rank] <jdl>
```

where `<jdl>` is the JDL file describing the job to be submitted. The `--rank` option also prints the rank of the CEs. The rank is a number expressing the adequacy of a CE (the higher, the better).

A JDL can specify the resources that it needs using the "Requirements" attribute, which is a Boolean ClassAd expression. To have a job scheduled to run on a given CE, this requirements expression must evaluate to true on the given CE. The evaluation is performed by the Workload Management System (WMS) during the match-making phase.

This is an example of a requirements expression:

```
Requirements = other.GlueCEInfoLRMSType == "PBS" &&  
other.GlueCEInfoTotalCPUs > 2 && Member("IDL1.7", other.GlueHostApplicationSoftwareRunTimeEnvironment);
```

The above expression requires a CE whose local resource manager is PBS, which has at least 2 CPUs, and where the IDL software version 1.7 is installed.